Міністерство освіти і науки України
Національний університет водного господарства та природокористування
Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки
Кафедра комп'ютерних технологій та економічної кібернетики

04-05-37

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних занять та самостійного вивчення навчальної дисципліни

«Математична логіка та теорія алгоритмів»

для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інформаційні системи та технології» спеціальності 126 «Інформаційні системи та технології» денної форми навчання

Рекомендовано науково-методичною радою з якості ННІ АКОТ
Протокол № 10 від 22.06.2020 р.

Рівне – 2020

Методичні вказівки до лабораторних занять та самостійного вивчення навчальної дисципліни «Математична логіка та теорія алгоритмів» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інформаційні системи та технології» спеціальності 126 «Інформаційні системи та технології» денної та заочної форм навчання [Електронне видання] / Карпович І. М., Гладка О. М. – Рівне : НУВГП, 2020. – 23 с.

Укладачі: Карпович І. М., к.ф.-м.н., доцент кафедри комп'ютерних технологій та економічної кібернетики; Гладка О. М., к.т.н., доцент кафедри комп'ютерних технологій та економічної кібернетики.

Відповідальний за випуск: Грицюк П. М., д.е.н., професор, завідувач кафедри комп'ютерних технологій та економічної кібернетики.

Керівник групи забезпечення
спеціальності Гладка О. М.

ЗМІСТ

Вступ

Методичні вказівки з дисципліни "Математична логіка та теорія алгоритмів" спрямовані на вивчення основних типів алгоритмічних моделей і передбачають елементи розробки та програмування їх інтерпретації мовою високого рівня. В циклі лабораторних робіт розглянуто основні поняття теорії алгоритмів і засоби реалізації основних алгоритмічних моделей.

Методичні вказівки призначені для вивчення студентами основних положень теорії алгоритмів та застосування алгоритмів у процесі розв'язання практичних завдань шляхом розроблення програмного забезпечення. Видання містить лише базові теоретичні відомості, необхідні для виконання лабораторних робіт, тому перед виконанням лабораторної роботи та під час підготовки до її захисту студентові необхідно ознайомитись з конспектом лекцій та опрацювати необхідний матеріал, наведений у переліку рекомендованої літератури, використовуючи також статті в інтернет-виданнях та актуальних наукових журналах з комп'ютерних технологій.

Для одержання оцінки з кожної роботи студент повинен у відповідності до наведених вимог розробити програмне застосування та оформити звіт, після чого продемонструвати на комп'ютері розроблену програму з виконанням усіх запропонованих викладачем тестів.

Усі завдання повинні виконуватись студентами індивідуально і не містити ознак плагіату як в оформленому звіті, так і в розробленому програмному забезпеченні.

Під час співбесіди на захисті лабораторної роботи студент повинен продемонструвати знання мети роботи, теоретичного матеріалу, методів виконання кожного етапу роботи, змісту основних розділів звіту з демонстрацією результатів на конкретних прикладах, практичних прийомів використання теоретичного матеріалу в розробленому програмному забезпеченні. Крім того, для успішного захисту роботи студент повинен відповісти на контрольні запитання, вміщені наприкінці опису відповідної роботи.

Лабораторна робота № 1

Тема: Купи та черги з пріоритетами. Пірамідальне сортування

Мета роботи

Вивчити основні концепції побудови наступних структур даних: куп та черг з пріоритетами. Навчитися обирати та реалізовувати структури даних для сортування, вставки, видалення та пошуку елементів. Навчитися реалізовувати та застосовувати алгоритм пірамідального сортування на практиці.

Основні відомості

Пірамідальне сортування ґрунтується на використанні спеціалізованих структур даних, що дозволяють керувати інформацією в процесі виконання алгоритму.

Для одного з поширених видів опрацювання даних – сортування – зручно використовувати такі структури даних, як черги з пріоритетами та купи.

Черга з пріоритетами – структура даних, яка зберігає множину елементів, у якій кожен елемент v має певний пріоритет із значенням key(v).

Двійкова купа (binary heap) або піраміда зручна для реалізації структури даних, що дозволяє швидко вставляти та вилучати елемент з максимальним пріоритетом (наприклад, максимальний за значенням).

Двійкова купа являє собою майже повне бінарне дерево, для якого виконується основна властивість купи: пріоритет кожної вершини вищий від пріоритетів її нащадків. В найпростішому випадку пріоритет кожної вершини можна вважати рівним її значенню. В такому випадку структура називається max-heap, оскільки корінь піддерева є максимумом зі значень елементів піддерева.

Алгоритм пірамідального сортування ґрунтується на використанні структури даних піраміда та складається з двох етапів: побудови піраміди та вилучення найбільших елементів.

Завдання для лабораторної роботи

1. Ознайомитися з літературою та основними теоретичними

відомостями, необхідними для виконання роботи.
2. Розробити програмне застосування, що виконує базові операції з купами та чергами з пріоритетами.
3. Розроблений програмний проект має складатися з окремих класів, що реалізують структури даних черга з пріоритетами та купа, а також має містити окремий модуль, що забезпечує інтерфейсну взаємодію з користувачем для роботи із створеними класами.
4. Клас, що реалізує чергу з пріоритетами, має дозволяти виконувати наступні операції на основі окремих методів: пошук елемента з найменшим ключем, видалення найменшого елемента, оновлення ключа елемента, вставлення елемента, видалення елемента, сортування елементів, виведення елементів на екран.
5. Клас, що реалізує купу, має дозволяти виконувати наступні операції на основі окремих методів: вставлення елемента, сортування елементів, побудова купи з невпорядкованого масиву, видалення елемента, сортування елементів із використанням купи, виведення елементів на екран.
6. Виконати тестування розробленого програмного забезпечення.
7. Розробити окремий модуль програмного забезпечення для реалізації пірамідального сортування на основі розроблених класів.
8. Розв’язати індивідуальне завдання за допомогою розробленої реалізації пірамідального сортування.
9. Порівняти одержані результати виконаних тестів, провести аналіз правильності, коректності та адекватності роботи розробленого програмного забезпечення.
10. Виконати порівняння пірамідального сортування з іншими видами сортування.
11. Оформити звіт.
12. Відповісти на контрольні запитання.

Зміст звіту

1. Мета роботи.
2. Завдання.

3. Текст розробленого програмного забезпечення з коментарями.
4. Результати роботи програмного забезпечення: таблиця тестових запусків, копії екранних форм.
5. Таблиця порівняння результатів застосування алгоритмів сортування.
6. Висновки, що відображають отримані результати виконання роботи, їх критичний аналіз.

Контрольні запитання

1. Що являє собою піраміда і яким властивостям така структура даних повинна задовольняти?
2. У яких випадках краще використовувати структуру даних типу купа, а у яких – чергу з пріоритетами?
3. Яким чином можна побудувати піраміду?
4. У чому полягає принцип додавання елемента у чергу з пріоритетами?
5. У чому полягає принцип сортування списку елементів із використанням купи?
6. Які способи реалізації черги з пріоритетами?
7. Які операції підтримуються в незростальній, а які в неспадній черзі з пріоритетами?
8. Які піраміди називаються незростальними, а які неспадними?
9. Чи є масив з відсортованими елементами неспадною пірамідою?
10. Сформулюйте особливості алгоритму пірамідального сортування.

Лабораторна робота № 2
Тема: Структури даних

Мета роботи

Засвоїти основні концепції геш-таблиць та бінарних дерев пошуку, зокрема, B-дерев. Навчитися використовувати геш-таблиці та B-дерева на практиці.

Основні відомості

Гешування ґрунтується на ідеї розподілу ключів у одновимірному масиві H[0..m-1], який називається геш-

таблицею. Розподіл виконується шляхом обчислення для кожного ключа значення деякої наперед визначеної функції h, яка називається геш-функцією. Геш-функція призначає кожному з ключів геш-адресу, яка являє собою ціле число від 0 до m-1. У загальному випадку геш-функція має задовольняти двом вимогам: 1) геш-функція має розподіляти ключі за комірками геш-таблиці якомога рівномірніше; 2) геш-функція повинна легко обчислюватися.

Методи побудови геш-функцій:

– метод ділення; – метод множення; – універсальне гешування.

Колізія – ситуація, коли два або більше ключів гешуються в одну й ту ж комірку геш-таблиці. Будь-яка схема гешування повинна мати механізм розв'язання колізій. Існує два основні підходи до опрацювання колізій: із застосуванням ланцюжків та із застосуванням адресації.

Бінарне дерево – скінченна множина вузлів, яка може бути або порожньою, або складатися з кореня та двох бінарних дерев, які не перетинаються та називаються лівим і правим піддеревами кореня.

Існують три алгоритми обходу бінарних дерев:

– у прямому порядку; – симетричний; - центрований.

B-дерева є узагальненням бінарних дерев пошуку. Вони являють собою збалансовані дерева пошуку, призначені для ефективної роботи з дисковою пам'яттю. Вузли B-дерев можуть мати багато дочірніх вузлів. У B-деревах всі записи даних або ключі зберігаються в листку у зростаючому порядку ключів. Батьківські вузли використовують для індексування.

У процесі пошуку в B-дереві, починаючи з кореня, рухаються ланцюжком покажчиків до листка, який може містити шуканий ключ. Після цього пошук відбувається серед ключів цього листка.

Завдання для лабораторної роботи

1. Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.
2. Розробити програмне застосування, що виконує базові операції з геш-таблицями та B-деревами.

3. Розроблений програмний проект має складатися з окремих класів, що реалізують структури даних, зокрема, геш-таблицю та бінарне дерево пошуку, а також має містити модуль, що забезпечує інтерфейсну взаємодію з користувачем для роботи із створеними класами.
4. Клас, що реалізує геш-таблицю, має дозволяти виконувати наступні операції на основі окремих методів: вставлення елемента, видалення елемента, пошук елемента, відображення структури геш-таблиці на основі використання параметрів, обраних за варіантом індивідуального завдання з п. 7.
5. Клас, що реалізує B-дерево, має дозволяти виконувати наступні операції на основі окремих методів: створення порожнього дерева, відображення структури дерева, пошук у дереві, вставлення ключа, видалення ключа.
6. Виконати тестування розробленого програмного забезпечення.
7. Розв'язати індивідуальне завдання за допомогою розроблених модулів програмного забезпечення.
8. Порівняти одержані результати виконаних тестів, провести аналіз правильності, коректності та адекватності роботи розробленого програмного забезпечення.
9. Оформити звіт.
10. Відповісти на контрольні запитання.

Зміст звіту

1. Мета роботи. 2. Завдання.
3. Текст розробленого програмного забезпечення з коментарями.
4. Результати роботи програмного забезпечення, що включають результати тестування та копії екранних форм.
5. Висновки, що відображають отримані результати виконання роботи, їх критичний аналіз.

Контрольні запитання

1. Що розуміють під гешуванням?
2. За яких умов слід використовувати геш-таблиці?
3. Що таке геш-функція та які висуваються вимоги до геш-функцій?

4. Які існують методи побудови геш-функцій?
5. Що таке колізія та які існують підходи для опрацювання колізій?
6. Що являє собою бінарне дерево?
7. Сформувати алгоритми обходу бінарних дерев.
8. Які операції можна виконувати з бінарними деревами пошуку?
9. Які дерева називають B-деревами? Якими властивостями вони характеризуються?
10. Які основні операції можна виконати з B-деревами?

Лабораторна робота № 3
Тема: Жадібні алгоритми

Мета роботи

Вивчити основні принципи та особливості жадібних алгоритмів. Навчитися використовувати жадібні алгоритми для розв'язання практичних завдань та обґрунтовувати прийняті рішення.

Основні відомості

Жадібний алгоритм – метод розв'язання оптимізаційних задач, який ґрунтується на тому, що процес прийняття рішень можна розбити на елементарні кроки, на кожному з яких приймається окреме рішення. Рішення, що приймається на кожному кроці, оптимальне лише на поточному кроці і повинно прийматися без урахування попередніх або наступних рішень. Іншими словами, на кожному кроці обирається найкращий варіант, вважаючи при цьому, що підсумкове рішення буде оптимальним.

Існують задачі, для яких послідовність таких жадібних виборів призведе до оптимального рішення для будь-якого примірника задачі, що розглядається. Однак для інших задач це твердження не виконується: для розв'язання таких задач жадібний алгоритм може використовуватися у випадку, якщо прийнятним є наближений розв'язок.

Коди Хаффмана широко використовуються для стиснення даних як доволі ефективний алгоритм. Алгоритм Хаффмана належить до жадібних алгоритмів і дозволяє закодувати текст, який побудований на основі n-символьного алфавіту. Кодування ґрунтується на принципі, за яким коротші коди призначаються символам, що зустрічаються частіше, а довші – символам, що зустрічаються рідше. Тобто коди символів мають змінну довжину. Такий підхід не дозволяє визначити, скільки бітів кодованого тексту представляє окремий символ. Тому в кодах Хаффмана для розв'язання цієї проблеми використовуються префіксні коди.

Завдання для лабораторної роботи

1. Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.
2. Розробити програмне застосування, що розв'язує задачу у відповідності з індивідуальним завданням з пункту 5 із використанням жадібного алгоритму.
3. Розроблений програмний проект має складатися з класу, що описує задачу, сформульовану в індивідуальному завданні, а також має містити окремий модуль, що забезпечує інтерфейсну взаємодію з користувачем.
4. Клас вхідних даних задачі має дозволяти:
– задавати початкові дані; – вводити нові параметри;
– коригувати та видаляти існуючі параметри;
– розв'язувати задачу з використанням жадібного алгоритму.
5. Отримати у викладача індивідуальне завдання.
6. Довести, що жадібний вибір для даної задачі є оптимальним розв'язком або принаймні є частиною деякого оптимального розв'язку.
7. Розробити програмне застосування, яке реалізує використання алгоритму Хаффмана для стискання даних текстового файлу у вигляді класу. Клас повинен мати методи, які дозволяють задати файл з даними, виконати стискання даних, визначити параметри виконаного стискання та зворотне перетворення, записати результати кодування/декодування у файл.

8. Виконати тестування розробленого у пп. 2 та 7 програмного забезпечення.
9. Порівняти одержані результати виконаних тестів, провести аналіз правильності, коректності та адекватності роботи розробленого програмного забезпечення.
10. Оформити звіт.
11. Відповісти на контрольні запитання.

Зміст звіту

1. Мета роботи.
2. Завдання.
3. Опис алгоритмів, за допомогою яких розв'язано індивідуальне завдання.
4. Обґрунтування можливості використання жадібних алгоритмів для розв'язання індивідуального завдання.
5. Текст розробленого програмного забезпечення з коментарями.
6. Результати роботи програмного забезпечення, що включають результати тестування та копії екранних форм.
7. Висновки, що відображають отримані результати виконання роботи та їх критичний аналіз.

Контрольні запитання

1. У чому суть жадібного вибору?
2. В яких випадках можна використовувати жадібні алгоритми?
3. Наведіть приклади задач, які можна розв'язувати з використанням жадібних алгоритмів.
4. Які задачі не можна розв'язувати за допомогою жадібних алгоритмів?
5. Які алгоритмічні методи є жадібними алгоритмами?
6. У чому полягає суть алгоритму Хаффмана?
7. Чи можна вважати алгоритм Хаффмана жадібним і чому?
8. Що являють собою префіксні коди?
9. Яким чином використовуються дерева під час створення кодів Хаффмана?

Лабораторна робота № 4.
Тема: Динамічне програмування

Мета роботи

Вивчити основні принципи динамічного програмування. Навчитися використовувати динамічне програмування для розв'язання практичних завдань.

Основні відомості

За своєю суттю динамічне програмування є методом проектування алгоритмів, що дозволяють розв'язувати задачі з підзадачами, які перекриваються. Такі підзадачі зазвичай виникають з рекурентних співвідношень, які зв'язують розв'язок даної задачі з розв'язками менших підзадач того ж виду. Замість того, щоб розв'язувати підзадачі, що перекриваються, знову і знову, динамічне програмування дозволяє розв'язати кожну з менших підзадач один раз, записуючи при цьому результат розв'язання в таблицю, з якої потім можна отримати розв'язання початкової задачі.

Завдання для лабораторної роботи

1. Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.
2. Розробити програмне застосування, що розв'язує задачу у відповідності з індивідуальним завданням з пункту 5 із використанням принципів динамічного програмування.
3. Розроблений програмний проект має складатися з класу, що описує задачу, сформульовану в індивідуальному завданні, а також має містити окремий модуль, що забезпечує інтерфейсну взаємодію з користувачем.
4. Клас вхідних даних задачі має дозволяти:
– задавати початкові дані; – вводити нові параметри;
– коригувати та видаляти існуючі параметри;
– розв'язувати відповідну задачу.
5. Отримати у викладача індивідуальне завдання.
6. Виконати аналіз розроблених алгоритмів для розв'язання індивідуального завдання щодо часу їх роботи та обсягу

використаної пам’яті.
7. Виконати тестування розробленого програмного забезпечення.
8. Порівняти одержані результати виконаних тестів, провести аналіз правильності, коректності та адекватності роботи розробленого програмного забезпечення і використаних методів.
9. Оформити звіт.
10. Відповісти на контрольні запитання.

Зміст звіту

1. Мета роботи.
2. Завдання.
3. Текст розробленого програмного забезпечення з коментарями.
4. Опис та аналіз часу роботи і необхідного обсягу пам’яті для роботи розроблених алгоритмів.
5. Результати роботи програмного забезпечення, що включають результати тестування та копії екранних форм.
6. Висновки, що відображають отримані результати виконання роботи та їх критичний аналіз.

Контрольні запитання

1. У чому суть динамічного програмування?
2. Для розв’язання яких задач можна використовувати динамічне програмування?
3. З яких етапів складається процес розроблення алгоритмів задач динамічного програмування?
4. Які існують види динамічного програмування і чим вони відрізняються?
5. У чому полягає суть перекриття підзадач? Наведіть приклади.
6. Яким чином можна реалізувати запам’ятовування в процесі динамічного програмування?
7. Яким чином можна поліпшити час роботи розробленого алгоритму задачі динамічного програмування?
8. Чи можна розв’язати завдання попередньої лабораторної роботи за допомогою динамічного програмування?

9. Яким чином побудувати граф підзадач для розробленого алгоритму?
10. Порівняйте жадібні алгоритми і алгоритми динамічного програмування.

Лабораторна робота № 5.

Тема: Алгоритми для роботи з графами. Алгоритми обходу графів

Мета роботи

Вивчити алгоритми обходу графів на основі пошуку в ширину та на основі пошуку в глибину. Навчитися застосовувати алгоритми обходу графів для розв'язання задач практики.

Основні відомості

Граф – це сукупність двох скінченних множин: множини точок та множини ліній, що попарно з'єднують деякі з цих точок. Множина точок формує вершини (вузли) графа. Множина ліній, що з'єднують вершини графа, формує ребра (дуги) графа. Поширеними способами є задання графів списком суміжних вершин або з допомогою матриці суміжності.

Для обходу графів існують два основні алгоритми: пошук в ширину та пошук в глибину.

Алгоритм пошуку в ширину в процесі обходу графа виконує обхід всіх вершин на відстані k перед тим, як перейти до пошуку вершин на відстані k+1.

Для відслідковування роботи алгоритму, пошук в ширину розфарбовує вершини графа в кольори: білий, сірий та чорний. Спочатку вершини мають білий колір. Після того, як під час пошуку вершина досягається в перший раз, вона фарбується у сірий колір. Далі в процесі пошуку вершина фарбується в чорний колір.

Пошук в глибину досліджує всі ребра, що виходять з вершини, яка була відкрита останньою, і залишає вершину тільки тоді, коли не залишається недосліджених ребер – в

такому випадку пошук повертається у вершину, з якої була досягнута дана вершина.

Аналогічно пошуку в ширину, пошук у глибину розфарбовує вершини графа в білий, сірий та чорний кольори.

Окрім безпосередньо обходу ребер графа та відвідування всіх вершин, алгоритми пошуку в глибину та ширину корисні під час дослідження ряду важливих властивостей графів.

Сильно зв'язним компонентом орієнтованого графа називають максимальну множину вершин даного графа, для кожної пари яких справедливо, що вони досяжні одна з одної. Для виконання пошуку сильно зв'язних компонент необхідно транспонувати заданий граф, тобто перетворити напрям ребер графа на протилежний.

Алгоритм пошуку сильно зв'язних компонент складається з двох пошуків у глибину:

– у початковому графі для обчислення часу завершення роботи з кожною вершиною;

– у транспонованому, де вершини розглядаються в порядку зменшення отриманих під час першого пошуку значень часу завершення роботи з кожною вершиною.

У такому випадку кожне дерево лісу пошуку в глибину, отримане під час другого пошуку, буде окремим сильно зв'язним компонентом.

Завдання для лабораторної роботи

1. Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.
2. Розробити програмне застосування, в якому реалізується алгоритм обходу графа на основі пошуку в глибину. Передбачити, що граф може бути як орієнтований, так і неорієнтований. В процесі пошуку має бути сформовано ліс пошуку в глибину. Для реалізації має використовуватися стек. Програмне застосування має бути побудовано на основі відповідного класу, який повинен дозволяти визначати граф, виконувати пошук в глибину, виводити побудований ліс пошуку в глибину, виводити результат обходу тощо.

3. Розробити програмне застосування, в якому реалізується алгоритм обходу графа на основі пошуку в ширину. Передбачити, що граф може бути як орієнтований, так і неорієнтований. В процесі пошуку має бути сформовано дерево пошуку в ширину. Для реалізації має використовуватися черга. Програмне застосування має бути побудовано на основі відповідного класу, який повинен дозволяти визначати граф, виконувати пошук в ширину, виводити побудоване дерево пошуку в ширину, виводити результат обходу тощо.
4. Виконати тестування розробленого програмного забезпечення.
5. Використовуючи розроблене програмне застосування, розробити окремі модулі, які використовуються для розв'язання задач у відповідності з індивідуальним завданням, отриманим у викладача, і реалізують інтерфейсну взаємодію користувача з програмою, та виконати відповідне тестування. Обґрунтувати вибір алгоритму обходу графа для кожної задачі, де це не задано явно.
6. Порівняти одержані результати виконаних тестів, провести аналіз правильності, коректності та адекватності роботи розробленого програмного забезпечення.
7. Оформити звіт.
8. Відповісти на контрольні запитання.

Зміст звіту

1. Мета роботи.
2. Завдання.
3. Опис та обґрунтування використання алгоритмів, за допомогою яких розв'язано індивідуальне завдання.
4. Текст розробленого програмного забезпечення з коментарями.
5. Результати роботи програмного забезпечення.
6. Висновки, що відображають отримані результати виконання роботи, їх критичний аналіз.

Контрольні запитання

1. Які існують способи зображення графів та яким чином вони пов'язані між собою?

2. Охарактеризуйте алгоритм обходу графа на основі пошуку в ширину.
3. Охарактеризуйте алгоритм обходу графа на основі пошуку в глибину.
4. Що являє собою дерево пошуку в ширину?.
5. Охарактеризуйте ліс пошуку в глибину. Як його сформувати?
6. Як розфарбовуються вершини графа під час його обходу?
7. Як від виду або способу зображення графа залежить ефективність алгоритмів пошуку в глибину і в ширину?
8. Чи є вимогливим до ресурсів алгоритм пошуку в ширину?
9. Яким чином під час реалізації в коді програми виконується повернення з тупикових вершин під час обходу графа?
10. Які структури даних використовуються для реалізації кожного алгоритму обходу графів?

Лабораторна робота № 6.

Тема: Алгоритми для роботи з графами. Алгоритми пошуку найкоротших шляхів

Мета роботи

Вивчити основні алгоритми пошуку найкоротшого шляху в графах: алгоритми Дейкстри, Флойда-Воршелла та Беллмана-Форда. Навчитися застосовувати алгоритми пошуку найкоротшого шляху в графі для розв'язання практичних задач.

Основні відомості

Алгоритм пошуку в ширину є алгоритмом пошуку найкоротшого шляху в незваженому графі, тобто в графі, кожному ребру якого відповідає одинична вага. Алгоритми Дейкстри, Беллмана-Форда та Флойда-Воршелла дозволяють працювати з графами, ребра яких мають не одиничну вагу.

Алгоритми пошуку найкоротших шляхів зазвичай ґрунтуються на твердженні, що найкоротший шлях між двома вершинами містить в собі інші найкоротші шляхи.

Алгоритм Дейкстри дозволяє виконувати пошук найкоротших шляхів з однієї вершини до всіх інших вершин у зваженому орієнтованому графі. Цей алгоритм знаходить

найкоротші шляхи до вершин графа в порядку їх віддаленості від вхідної вершини, тобто спочатку знаходиться найкоротший шлях від вхідної вершини до найближчої, потім до другої найближчої і т. д. Таким чином, перед початком і-ої ітерації алгоритм визначає найкоротші шляхи до (і-1)-ої вершин, найближчих до вхідної. Вказані вершини, вхідна вершина та ребра найкоротших шляхів утворюють піддерево графа.

Чергова найближча до вхідної вершина може бути знайдена серед вершин, суміжних з отриманим піддеревом: для кожної суміжної вершини обчислюється сума відстаней до найближчої вершини дерева, після чого обирається вершина з найменшою сумою.

Алгоритм Беллмана-Форда дозволяє виконувати пошук найкоротших шляхів з однієї вершини до всіх інших вершин у випадку, коли вага кожного з ребер може бути від'ємною. Алгоритм повертає логічне значення, яке вказує, чи міститься в графі цикл з від'ємною вагою, що досягається з джерела. Якщо такий цикл існує, то алгоритм вказує на те, що розв'язку не існує. Якщо таких циклів не існує, то алгоритм повертає найкоротші шляхи та їх ваги.

Алгоритм Флойда-Воршелла призначений для пошуку найкоротшого шляху між всіма парами вершин. Цей алгоритм може бути реалізовано за допомогою обчислення матриці відстаней зваженого графа шляхом послідовних обчислень. Кожна матриця на відповідному кроці містить довжини найкоротших шляхів між будь-якими двома вершинами, при чому кількість проміжних вершин у них не може перевищувати відповідний номер поточного кроку.

Завдання для лабораторної роботи

1. Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.
2. Розробити програмне застосування, в якому реалізується алгоритм Дейкстри на основі виділення відповідного класу для виконання всіх необхідних обчислень, визначення параметрів (в тому числі безпосередньо визначення графа) та отримання результатів.

3. Розширити функціональність розробленого програмного забезпечення за допомогою реалізації алгоритму Флойда-Воршелла на основі відповідного класу, який повинен забезпечувати виконання операцій, аналогічних п. 2.
4. Розширити функціональність розробленого програмного забезпечення за допомогою реалізації алгоритму Беллмана-Форда на основі відповідного класу, який повинен забезпечувати виконання операцій, аналогічних п. 2.
5. Виконати тестування розробленого програмного забезпечення.
6. Використовуючи розроблене програмне застосування, розробити окремі модулі, які використовуються для розв'язання задач індивідуального завдання і реалізують інтерфейсну взаємодію користувача з програмою, та виконати відповідне тестування. Вибір методів розв'язання задач обґрунтувати (якщо можна застосувати декілька методів, то застосувати їх).
7. Порівняти одержані результати виконаних тестів, провести аналіз правильності, коректності та адекватності роботи розробленого програмного забезпечення.
8. Оформити звіт.
9. Відповісти на контрольні запитання.

Зміст звіту

1. Мета роботи.
2. Завдання.
3. Опис та обґрунтування використання алгоритмів, за допомогою яких розв'язано індивідуальне завдання.
4. Текст розробленого програмного забезпечення з коментарями.
5. Результати роботи програмного забезпечення.
6. Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз.

Контрольні запитання

1. Для роботи з якими видами графів призначені алгоритми пошуку найкоротших шляхів?
2. За яких умов може використовуватися алгоритм Дейкстри?
3. Охарактеризувати алгоритм Дейкстри.

4. На основі яких структур даних побудований алгоритм Дейкстри?
5. Для розв'язання яких задач призначений алгоритм Дейкстри?
6. Для якого виду графів може застосовуватися алгоритм Флойда-Воршелла?
7. Охарактеризуйте алгоритм Флойда-Воршелла.
8. У яких випадках застосовують алгоритм Беллмана-Форда?
9. Охарактеризуйте алгоритм Беллмана-Форда.
10. Метод послаблення та його використання.

Лабораторна робота № 7.

Тема: Моделювання транспортної мережі за допомогою графів

Мета роботи

1. Вивчити метод Форда-Фалкерсона для розв'язання задачі про максимальний потік у транспортній мережі. Навчитися застосовувати метод Форда-Фалкерсона для розв'язання практичних завдань.

Основні відомості

Під час моделювання транспортної мережі за допомогою графів кожне орієнтоване ребро можна розглядати як деякий канал, яким пересувається певний продукт. Кожний канал має задану пропускну здатність, що характеризує максимальну швидкість пересування продуктів каналом. Вершини є точками перетину каналів. Через вершини, відмінні від джерела та стоку, продукт просувається, не накопичуючись.

Задача про максимальний потік полягає у знаходженні такого потоку в транспортній мережі, що сума потоків з витоку (до стоку) є максимальною.

Метод Форда-Фалкерсона дозволяє розв'язати задачу про максимальний потік шляхом ітеративного збільшення значення потоку. Спочатку потік обнуляється, а далі на кожній ітерації величина потоку збільшується шляхом пошуку збільшувального шляху.

Завдання для лабораторної роботи

1. Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.
2. Розробити програмне застосування, в якому реалізується алгоритм Форда-Фалкерсона на основі виділення відповідного класу для виконання всіх необхідних обчислень, визначення параметрів (в тому числі, безпосередньо визначення графа) та отримання результатів.
3. Виконати тестування розробленого програмного забезпечення.
4. Використовуючи розроблене програмне застосування, розв'язати задачу у відповідності з індивідуальним завданням, реалізуючи інтерфейсну взаємодію користувача з програмою, та виконати відповідне тестування.
5. Порівняти одержані результати виконаних тестів, провести аналіз правильності, коректності та адекватності роботи розробленого програмного забезпечення.
6. Оформити звіт.
7. Відповісти на контрольні запитання.

Зміст звіту

1. Мета роботи.
2. Завдання.
3. Текст розробленого програмного забезпечення з коментарями.
4. Результати роботи програмного забезпечення.
5 Висновки, що відображають отримані результати виконання роботи та їх аналіз.

Контрольні запитання

1. Поняття транспортної мережі
2. Що являє собою потік у транспортній мережі?
3. Яким властивостям повинен задовольняти потік у мережі?
4. Які шляхи називаються збільшувальними у транспортній мережі?
5. Теорема Форда-Фалкерсона
6. Поняття залишкової мережі
7. Що являє собою розріз транспортної мережі?

8. Який розріз транспортної мережі називають мінімальним?
9. Що таке паросполучення?
10. З яких кроків складається алгоритм Форда-Фалкенсона?

ЛІТЕРАТУРА

1. Нікітченко М. С. Математична логіка та теорія алгоритмів. Київ : ВПЦ “Київський університет”, 2008. 528 с.
2. Клакович Л. М., Левицька С. М., Костів О. М. Теорія алгоритмів. Львів : Вид. Львів. ун-ту, 2008. 154 с.
3. Алгоритмы: построение и анализ / Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л Ривест, Клиффорд Штайн. Пер. с англ. М. : ООО “И.Д. Вильямс”, 2013. 1328 с.
4. Левитин А. В. Алгоритмы: введение в разработку и анализ. Пер. с англ. М. : ООО “И.Д. Вильямс”, 2006. 576 с.
5. Ахо А. В., Хопкрофт Д. Э., Ульман Д. Д. Структуры данных и алгоритмы : уч. пос. / пер. с англ. М. : ООО “И. Д. Вильямс”, 2007. 400 с.
6. Кнут Д. Э. Искусство программирования. Том 1. Основные алгоритмы; Том 3. Сортировка и поиск. М. : Вильямс, 2012. 832 с.
7. Кормен, Т. Х. Алгоритмы. Вводный курс. М. : Вильямс, 2014. 208 с.
8. Kleinberg, J. Algorithm Design. Boston: Pearson, 2005. 864 p.
9. Mehlhorn, K., Sanders, P. Algorithms and Data Structures. Berlin : Springer, 2008. 300 p.